Analyze bulk RNAseq data

Lev Litichevskiy

December 15, 2022

Necessary input files:

- Counts matrix: This file should have genes in the rows and samples in the columns. Each value is the number of times that gene X was detected in sample Y.
- Metadata: This file should have important experimental metadata, with each row corresponding to a sample and the columns corresponding to relevant experimental variables.

I am using a demo dataset with 200 genes and 14 samples. The samples are bulk RNAseq of colonic tissue from mice treated with the antibiotics neomycin (n=5) or vancomycin (n=5) and control mice that were not treated with antibiotics (n=4).

Search the document for "CHANGEME" to see which lines of code you will need to change for your specific project. This is not comprehensive – other lines may require changing!

A very nice companion to this analysis is the DESeq2 tutorial. This RNAseq workflow vignette by Michael Love et al. has a lot of overlap with the DESeq2 tutorial but other potentially helpful details.

Load packages

You'll probably need to download some of these packages before you can use them.

```
library(tidyverse)
library(DESeq2)

# this is just how I like my plots
theme_set(theme_bw(base_size=12))
```

Import metadata

```
# CHANGEME: to your metadata file
meta.df <- read.table("demo_rnaseq_metadata.tsv", sep="\t", header=T)
```

Import data

```
# CHANGEME: to your actual data matrix
counts.df <- read.table("demo_rnaseq_counts_matrix.tsv", sep="\t", header=T) %>%

  # move gene_name to be the rownames
  # CHANGEME: if your gene have a different column name
  column_to_rownames("gene_name")
```

Convert to integer

DESeq2 wants the counts matrix to be integers, so we'll round our data to the nearest integer and convert the data type to be "integer" rather than "double".

```
# round to the nearest integer
counts.mat.int <- counts.df %>%
  as.matrix() %>%
  round()

# convert data type to integer
mode(counts.mat.int) <- "integer"

# convert matrix back to a df
counts.df.int <- data.frame(counts.mat.int)
```

Verify that columns of the counts matrix match the rows of the metadata

```
# CHANGEME: this assumes that the metadata column corresponding to sample IDs is
# called `sample.ID`; change this to whatever column in your metadata matches
# the columns of your counts matrix
if (all(colnames(counts.df.int) == meta.df$sample.ID)) {

  cat("All good.\n")

} else {

  cat("The columns of your counts matrix don't match your metadata.\n")
  cat("First 3 samples in counts matrix:", colnames(counts.df.int)[1:3], "\n")
  cat("First 3 samples in metadata:", meta.df$sample.ID[1:3], "\n")

}
```

```
## All good.
```

Create DESeq2 object

The `design` argument in the following code chunk is very important. This is where you tell DESeq2 about the design of your experiment. See the DESeq2 tutorial for more information.

```
dds <- DESeqDataSetFromMatrix(
  countData=counts.df.int,
  colData=meta.df,

  # CHANGEME: this is where you tell DESeq2 about your experiment
  # each of the words after the ~ should correspond to columns in meta.df
  # in the demo dataset, we only have antibiotic status, but an example of
  # another design formula is something like ~ sex + age + timepoint
  design= ~ abx)
```

```
## Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
## design formula are characters, converting to factors
```

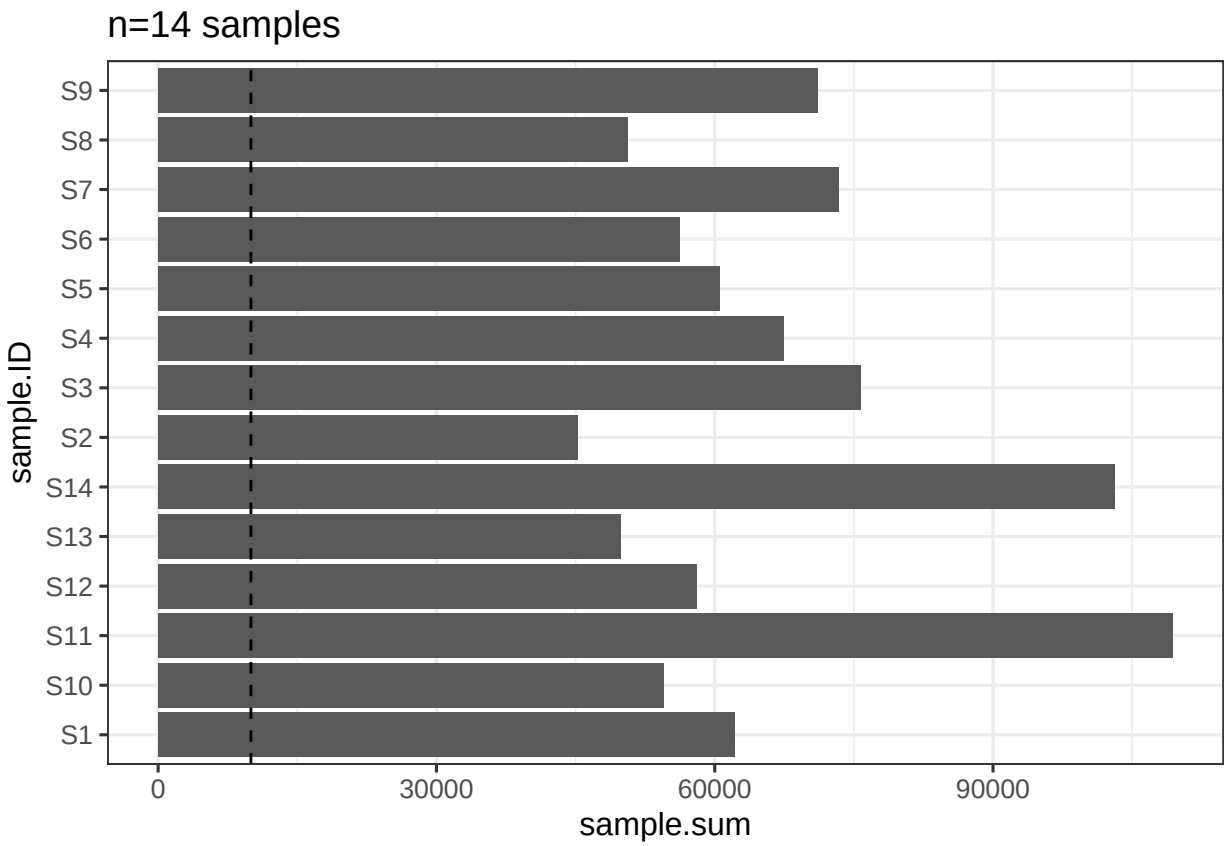
Don't worry if you see "Warning: some variables in design formula are characters, converting to factors".

Filtration

We want to exclude samples with very few total counts and genes that were detected very rarely. These thresholds depend a lot on your specific experiment. As a rough rule of thumb, I like samples to have at least 5M total counts, and I might discard genes that receive fewer than 100 counts across all samples. The gene filtering isn't super important because other downstream analyses also filter out low-abundance genes.

Remove samples with very low sequencing depth

```
# CHANGEME if you want to change this filtering threshold
SAMPLE.SUM.THRESHOLD <- 10000 #5000000

data.frame(sample.sum=colSums(counts(dds))) %>%
  rownames_to_column("sample.ID") %>%
  ggplot(aes(y=sample.ID, x=sample.sum)) +
  geom_col() +
  geom_vline(xintercept=SAMPLE.SUM.THRESHOLD, lty=2) +
  labs(title=sprintf("n=%i samples", ncol(dds)))
```



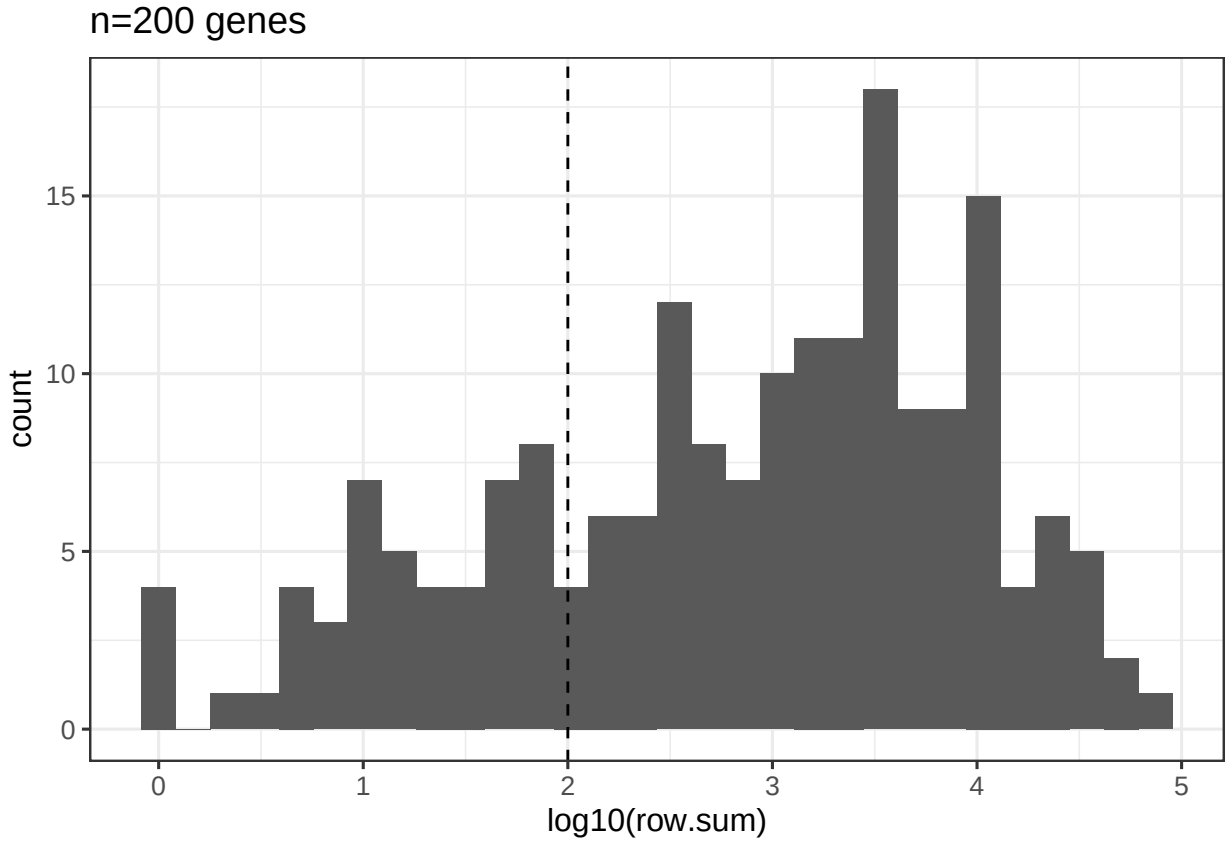
- We have many fewer than 5M counts per sample because we're using a small demo dataset
- Here, I set a threshold of 10k counts per sample

Remove genes with very low expression

```
# CHANGEME if you want to change this filtering threshold
GENE.SUM.THRESHOLD <- 100

data.frame(row.sum=rowSums(counts(dds))) %>%
  ggplot(aes(log10(row.sum))) +
  geom_histogram() +
  geom_vline(xintercept=log10(GENE.SUM.THRESHOLD), lty=2) +
  labs(title=sprintf("n=%i genes", nrow(dds)))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 8 rows containing non-finite values (stat_bin).
```



Do filtering

```
samples.to.keep <- colnames(counts(dds))[colSums(counts(dds)) > SAMPLE.SUM.THRESHOLD]
genes.to.keep <- rownames(counts(dds))[rowSums(counts(dds)) > GENE.SUM.THRESHOLD]
dds.filt <- dds[genes.to.keep, samples.to.keep]
```

```
dim(dds)
```

```
## [1] 200  14
```

```
dim(dds.filt)
```

```
## [1] 142  14
```

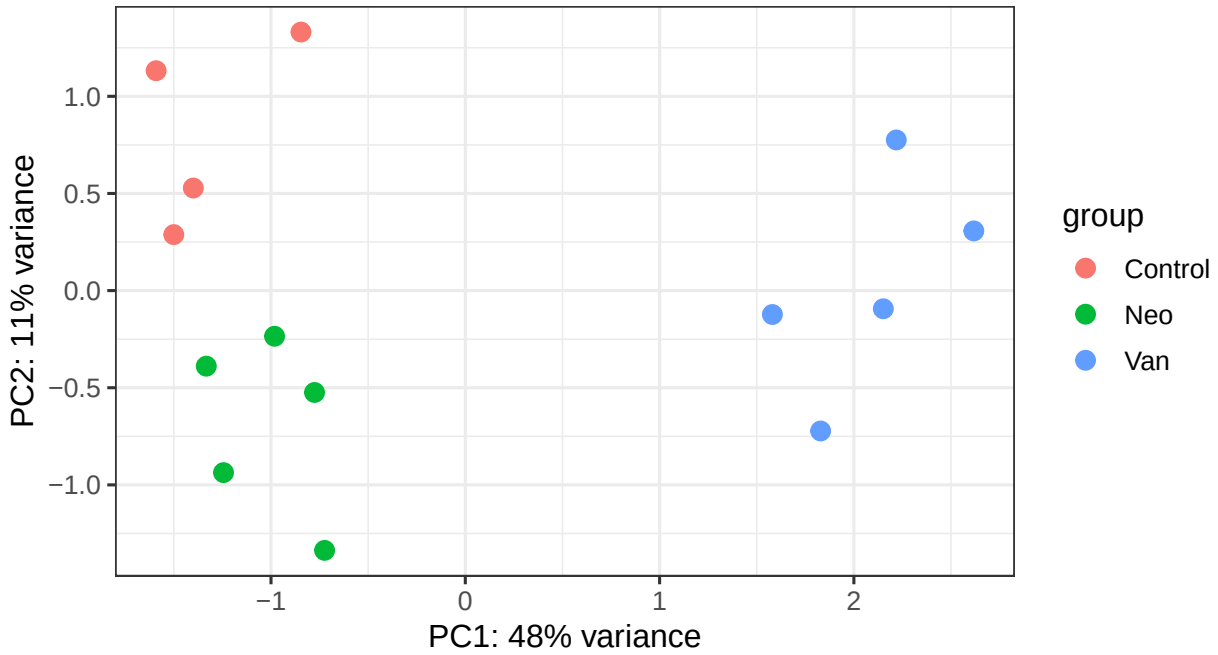
- 200 genes -> 142 genes
- No filtration of samples

PCA

Okay, now we're on to real analysis. To make a PCA plot, we first apply the variance-stabilizing transformation (VST). Other transformations, like log2cpm or rlog (available through DESeq2), would also be fine. The two things that VST is adjusting for are 1) library size, and 2) ensuring that highly expressed genes don't dominate over lowly expressed genes. See this section in the RNAseq workflow vignette for more information.

```
# CHANGEME: because our demo dataset has very few genes, we set nsub to 100, but
# you should use more genes than this; the default is 1000
vsd <- vst(dds.filt, nsub=100)
```

```
# CHANGEME: set intgroup to your experimental variable of interest
plotPCA(vsd, intgroup="abx")
```



- You can add any other ggplot `geom` functions to this plot
- See the "Principal component plot of the samples" section in the DESeq2 tutorial about how to do this.

Find differentially expressed genes

Differential expression is calculated based on the design formula you provided earlier.

Importantly, you provide the raw counts to DESeq2, not transformed counts, because DESeq2 uses raw counts for its internal modeling.

```
dds.filt <- DESeq(dds.filt)
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
## fitting model and testing
```

See possible comparisons

```
resultsNames(dds.filt)
```

```
## [1] "Intercept"          "abx_Neo_vs_Control" "abx_Van_vs_Control"
```

- These are the possible options you can provide to the "name" argument of the `results` function below

Neo v. control

In this case, both antibiotics are compared to control, which is what we want. But often, you end up with comparisons against some group that isn't actually the control or reference group. See this note in the DESeq2 tutorial about how to get around this.

```
# CHANGEME: "name" should be one of the outputs of resultsNames above
neo.deseq.df <- results(dds.filt, name="abx_Neo_vs_Control") %>%

  # convert to dataframe for convenience
  as.data.frame()
```

```
neo.deseq.df %>% dplyr::filter(padj < 0.01) %>% nrow
```

```
## [1] 4
```

```
neo.deseq.df %>% dplyr::filter(padj < 0.001) %>% nrow
```

```
## [1] 1
```

```
neo.deseq.df %>% dplyr::filter(padj < 0.0001) %>% nrow
```

```
## [1] 1
```

- We have a very small number of differentially expressed genes (DEGs) because we are using a demo dataset
  - padj < 0.01: 4 genes
  - padj < 0.001: 1 gene
  - padj < 0.0001: 1 gene

Display top DEGs

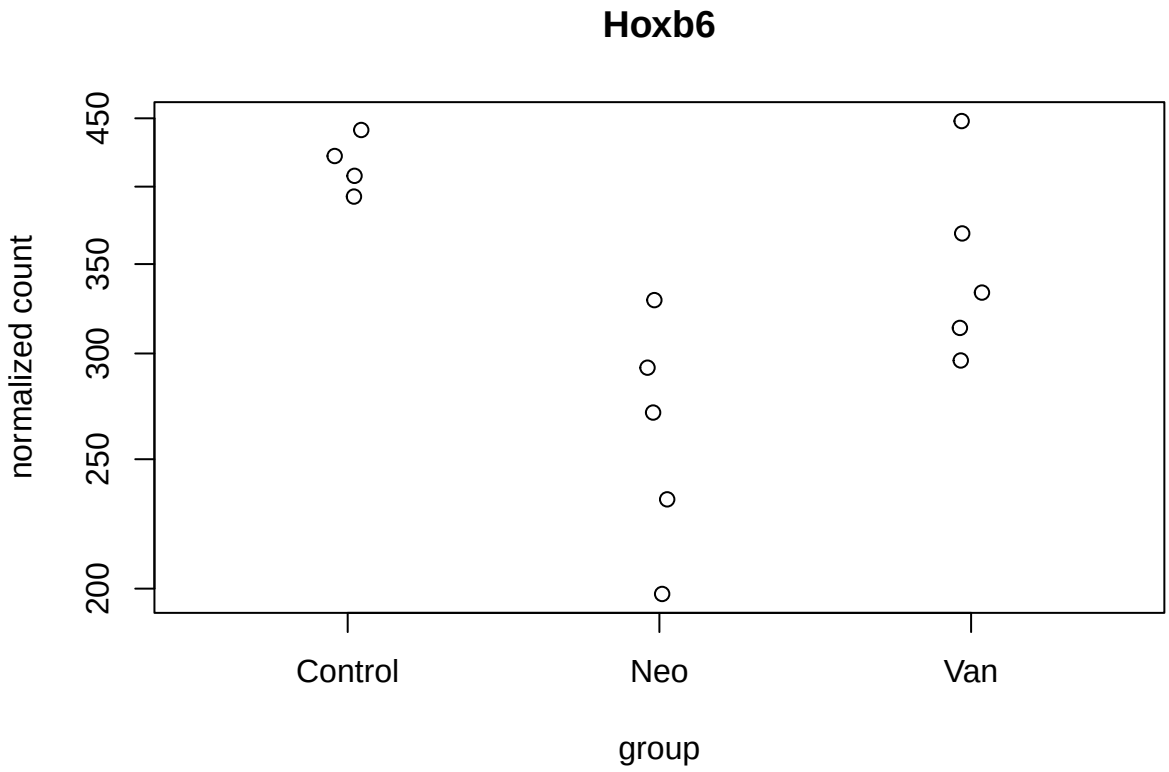
```
neo.deseq.df %>%
  slice_min(padj, n=10, with_ties=F) %>%
  rownames_to_column("gene_name") %>%
  dplyr::select(gene_name, log2FoldChange, pvalue, padj) %>%
  arrange(padj)
```

```
##    gene_name log2FoldChange       pvalue         padj
## 1      Hoxb6     -0.6515914 6.369943e-07 6.561041e-05
## 2     Slc7a7      0.3114003 3.612509e-05 1.860442e-03
## 3       Sox9     -0.5180119 2.208700e-04 7.583203e-03
## 4     Myo18a      0.2681451 3.574927e-04 9.205438e-03
## 5       Cav2      0.7102901 1.090607e-03 1.969479e-02
## 6      Pparg      0.4731417 1.147269e-03 1.969479e-02
## 7      Slfn4      0.4959961 2.501551e-03 3.680854e-02
```

```
## 8        Gcg     -0.3754538 3.920975e-03 5.038109e-02
## 9     Zfp512b      0.3720786 4.402232e-03 5.038109e-02
## 10     Oxa1l     -0.2443161 6.656590e-03 6.856287e-02
```

Look at one gene

```
# CHANGEME: to your gene of interest and experimental group of interest
plotCounts(dds.filt, gene="Hoxb6", intgroup="abx")
```



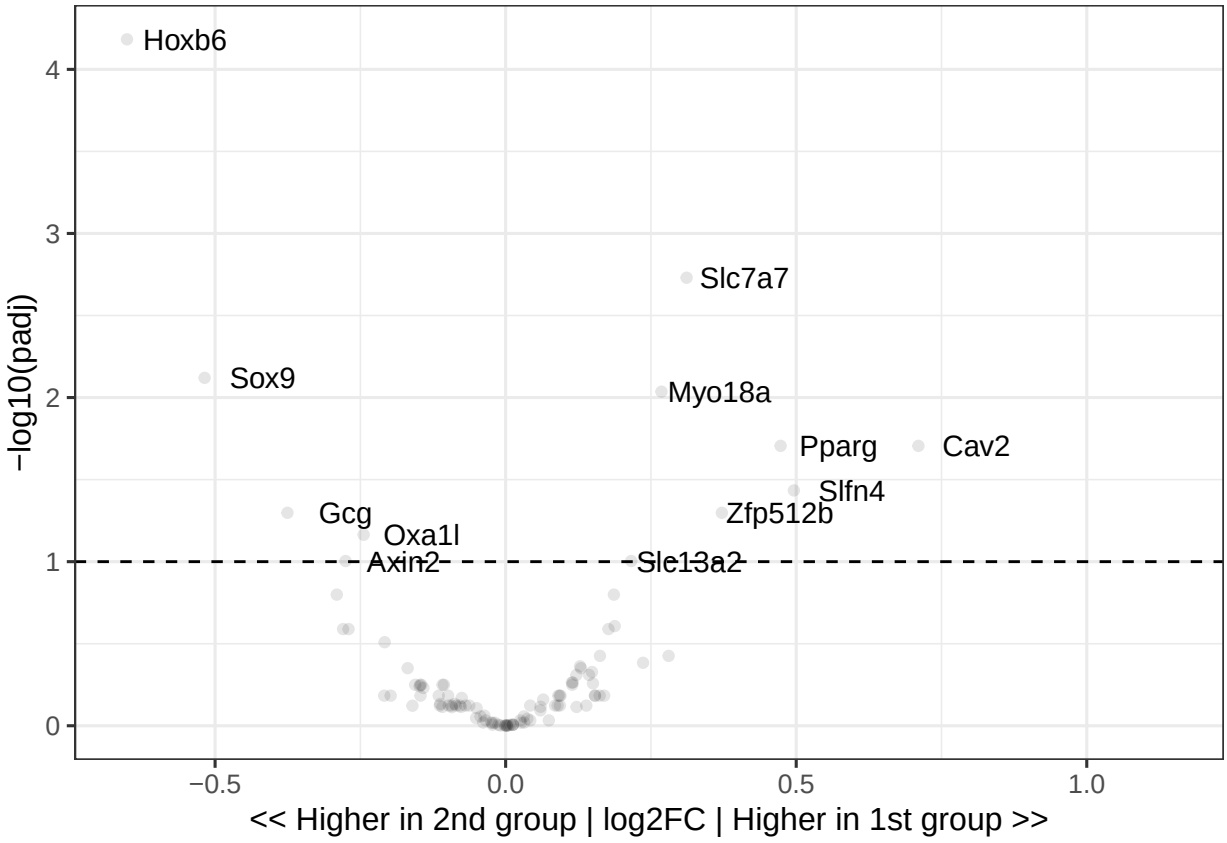
Volcano plot

```
# CHANGEME: change to what adjusted p-value threshold you want
# 0.01 is a rule of thumb, I'm using 0.1 to demonstrate labeling
PADJ.THRESHOLD <- 0.1 # 0.01

neo.deseq.df %>%
  rownames_to_column("gene_name") %>%

  # only label DEGs
  mutate(label=case_when(
    padj < PADJ.THRESHOLD ~ gene_name,
    TRUE ~ NA_character_
  )) %>%

  ggplot(aes(x=log2FoldChange, y=-log10(padj), label=label)) +
  geom_point(alpha=0.1) +
  geom_hline(yintercept=-log10(PADJ.THRESHOLD), lty=2) +
  geom_text(nudge_x=0.1) +
```

```
  # CHANGEME: improve this label with what your first and second groups actually are
  labs(x="<< Higher in 2nd group | log2FC | Higher in 1st group >>")
```

```
## Warning: Removed 39 rows containing missing values (geom_point).
```

```
## Warning: Removed 130 rows containing missing values (geom_text).
```



Vanc v. control

```
# CHANGEME: "name" should be one of the outputs of resultsNames above
vanc.deseq.df <- results(dds.filt, name="abx_Van_vs_Control") %>%

  # convert to dataframe for convenience
  as.data.frame()
```

And then repeat the same steps that we did for neo versus control.

Pathway enrichment

It's possible to do pathway enrichment in R, but perhaps the quickest and easiest thing is to export your DEGs and provide them to Enrichr.

```
neo.deseq.df %>% dplyr::filter(padj < PADJ.THRESHOLD) %>%
  rownames %>% cat
```

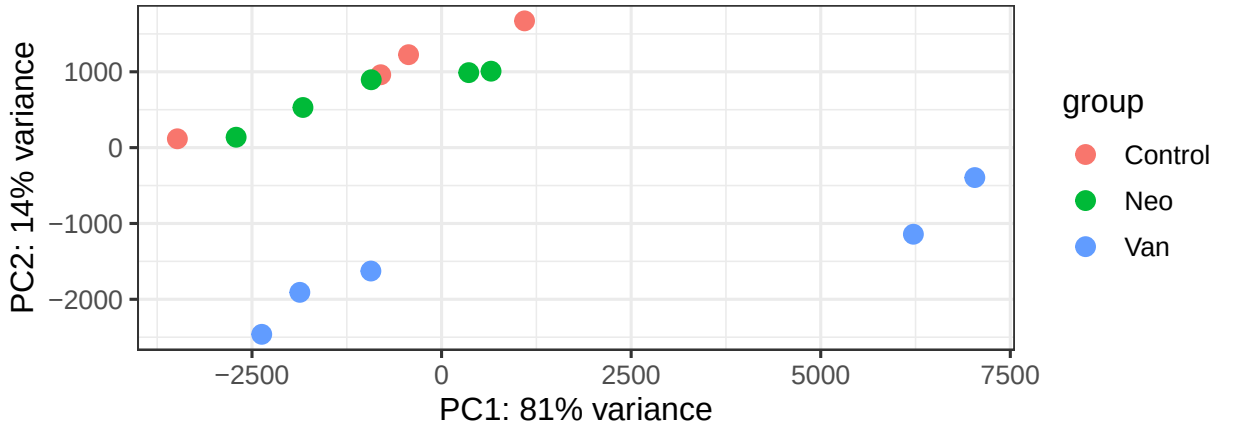
```
## Cav2 Axin2 Slfn4 Gcg Pparg Sox9 Myo18a Hoxb6 Zfp512b Slc7a7 Oxa1l Slc13a2
```

Appendix: data normalization

Raw counts not appropriate for PCA

Because PCA will be sensitive to sequencing depth.

```
plotPCA(DESeqTransform(dds), intgroup="abx")
```



- We see that the vancomycin group is split into two groups

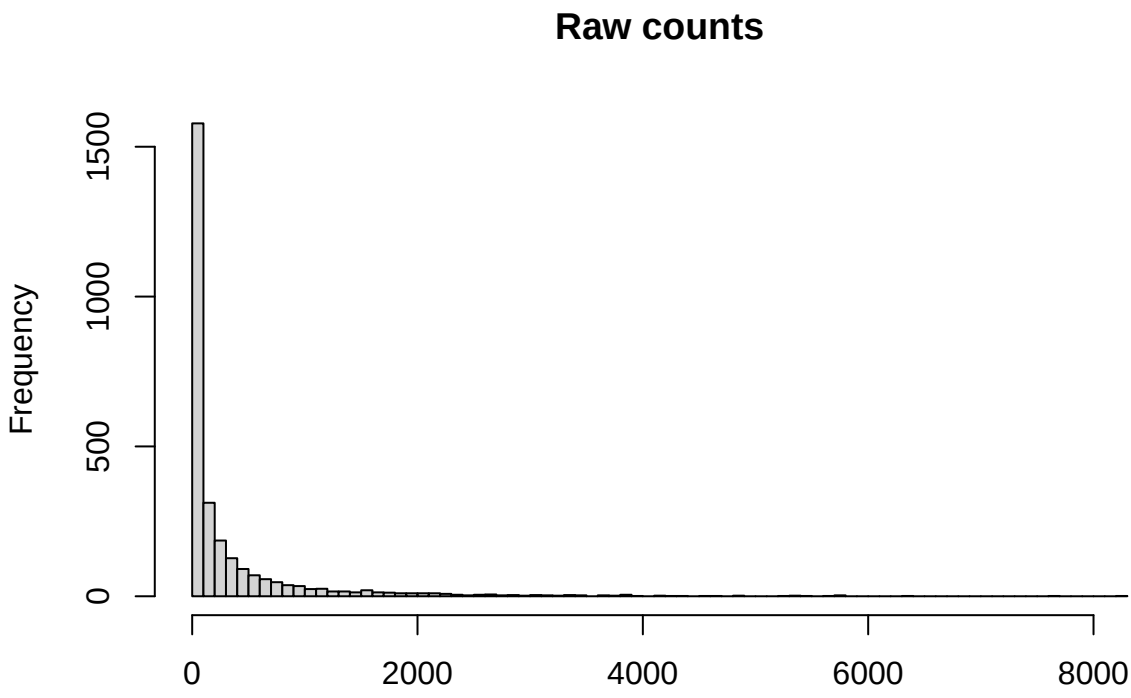
```
plotPCA(DESeqTransform(dds), intgroup="abx", returnData=T) %>%
  merge(data.frame(sample.sum=colSums(counts(dds))), by="row.names") %>%
  dplyr::filter(abx == "Van") %>%
  dplyr::select(sample.sum, PC1, PC2) %>%
  arrange(PC1)
```

```
##   sample.sum        PC1        PC2
## 1      49875 -2371.4184 -2461.9766
## 2      54514 -1869.1349 -1908.4514
## 3      58083  -932.2249 -1627.3218
## 4     103104  6221.8594 -1142.0560
## 5     109404  7031.9590  -394.8798
```

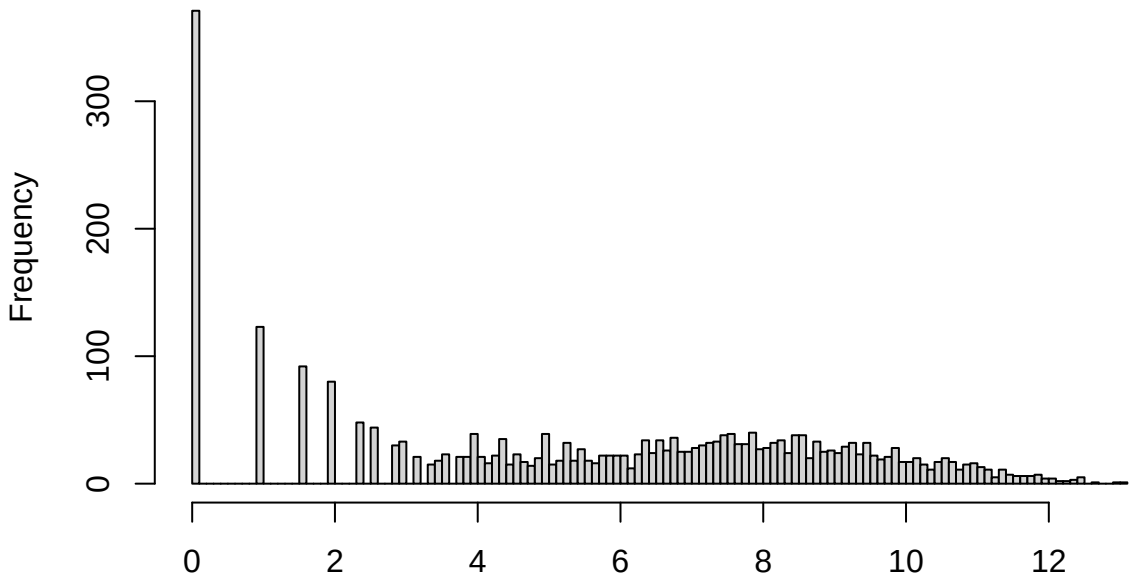
- The two samples off to the right have twice the sequencing depth of the other 3 vancomycin samples

Log transformation helps make the data look more normal

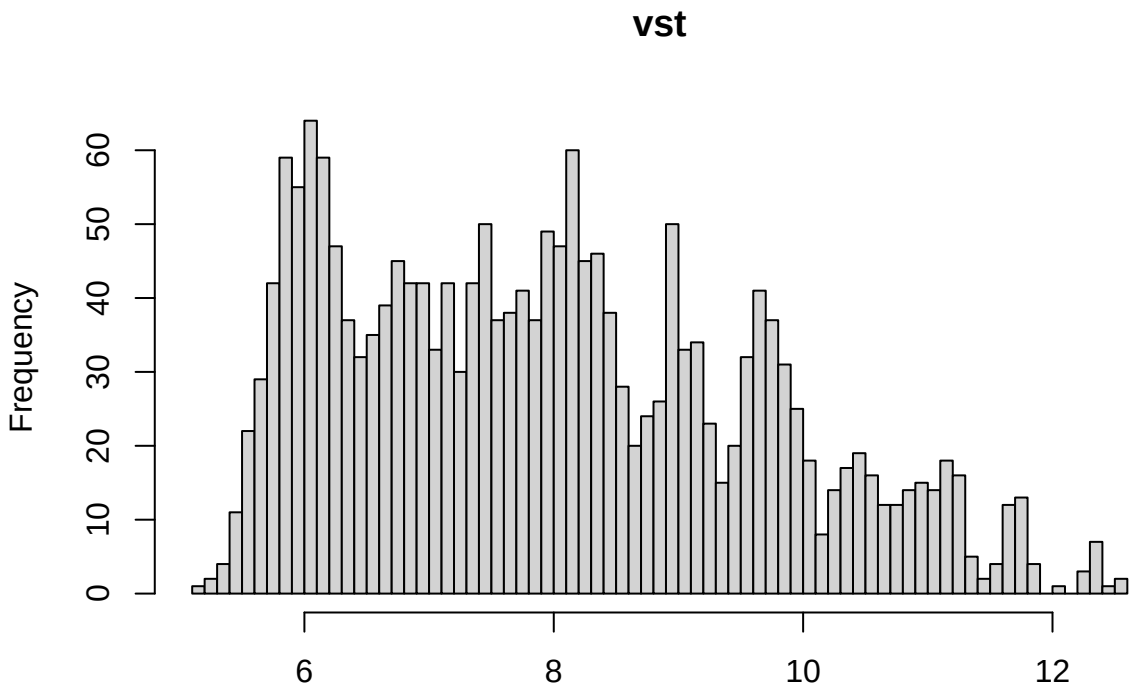
```
as.numeric(counts(dds)) %>%
  hist(breaks=100, xlab="", main="Raw counts")
```




```
as.numeric(log2(counts(dds) + 1)) %>%
  hist(breaks=100, xlab="", main="log2(count+1)")
```




```
as.numeric(assay(vsd)) %>%
  hist(breaks=100, xlab="", main="vst")
```



- We see that vst dealt better with the zeros